

APPENDIX

An Overview of the Hierarchical Data Model

This appendix provides an overview of the hierarchical data model.¹ There are no original documents that describe the hierarchical model, as there are for the relational and network models. The principles behind the hierarchical model are derived from Information Management System (IMS), which is the dominant hierarchical system in use today by a large number of banks, insurance companies, and hospitals as well as several government agencies. Another popular hierarchical DBMS is MRI's System-2000 (which was later sold by SAS Institute).

In this appendix we present the concepts for modeling hierarchical schemas and instances, the concept of a virtual parent-child relationship, which is used to overcome the limitations of pure hierarchies, and the constraints on the hierarchical model. A few examples of data manipulation are included.

1. The complete chapter on the hierarchical data model and the IMS system from the second edition of this book is available at the Web site for the book. This appendix is an edited excerpt of that chapter.

D.1 Hierarchical Database Structures

D.1.1 Parent-Child Relationships and Hierarchical Schemas

The hierarchical model employs two main data structuring concepts: records and parent-child relationships. A **record** is a collection of **field values** that provide information on an entity or a relationship instance. Records of the same type are grouped into **record types**. A record type is given a name, and its structure is defined by a collection of named **fields** or **data items**. Each field has a certain data type, such as integer, real, or string.

A **parent-child relationship type (PCR type)** is a 1:N relationship between two record types. The record type on the 1-side is called the **parent record type**, and the one on the N-side is called the **child record type** of the PCR type. An **occurrence** (or **instance**) of the PCR type consists of *one record* of the parent record type and a number of records (zero or more) of the child record type.

A **hierarchical database schema** consists of a number of hierarchical schemas. Each **hierarchical schema** (or **hierarchy**) consists of a number of record types and PCR types.

A hierarchical schema is displayed as a **hierarchical diagram**, in which record type names are displayed in rectangular boxes and PCR types are displayed as lines connecting the parent record type to the child record type. Figure D.1 shows a simple hierarchical diagram for a hierarchical schema with three record types and two PCR types. The record types are DEPARTMENT, EMPLOYEE, and PROJECT. Field names can be displayed under each record type name, as shown in Figure D.1. In some diagrams, for brevity, we display only the record type names.

We refer to a PCR type in a hierarchical schema by listing the pair (parent record type, child record type) between parentheses. The two PCR types in Figure D.1 are (DEPARTMENT, EMPLOYEE) and (DEPARTMENT, PROJECT). Notice that PCR types *do not* have a name in the hierarchical model. In Figure D.1 each *occurrence* of the (DEPARTMENT, EMPLOYEE) PCR type relates one department record to the records of the *many* (zero or more) employees who work in that department. An *occurrence* of the (DEPARTMENT, PROJECT) PCR type relates a department record to the records of projects controlled by that department. Figure D.2 shows two PCR occurrences (or instances) for each of these two PCR types.

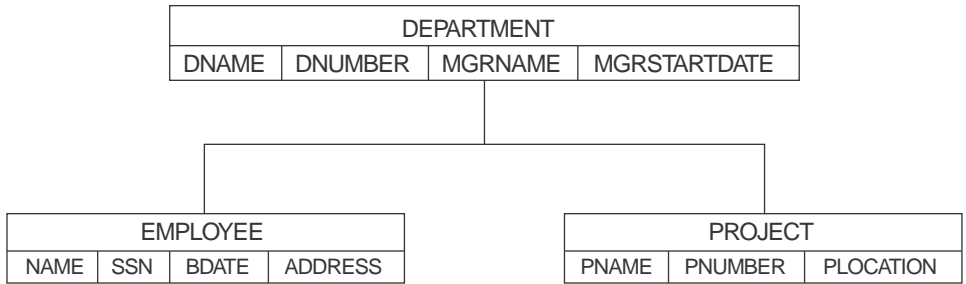


Figure D.1 A hierarchical schema.

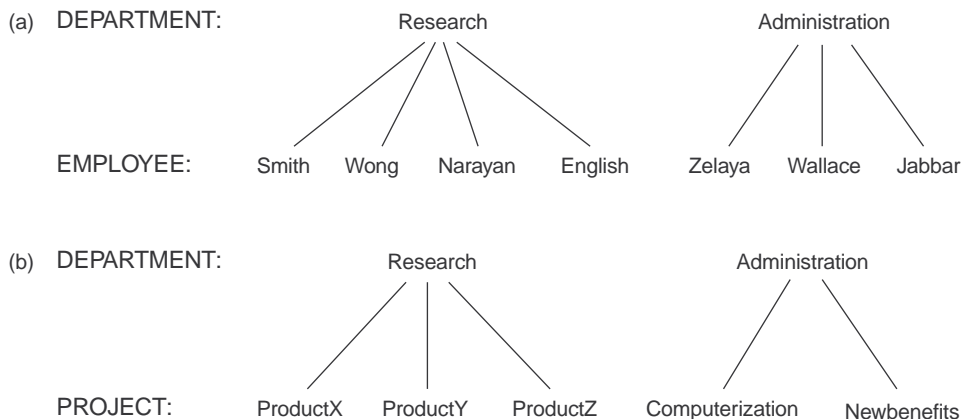


Figure D.2 Occurrences of Parent-Child Relationships. (a) Two occurrences of the PCR type (DEPARTMENT, EMPLOYEE). (b) Two occurrences of the PCR type (DEPARTMENT, PROJECT).

D.1.2 Properties of a Hierarchical Schema

A hierarchical schema of record types and PCR types must have the following properties:

1. One record type, called the **root** of the hierarchical schema, does not participate as a child record type in any PCR type.
2. Every record type except the root participates as a child record type in *exactly one* PCR type.
3. A record type can participate as parent record type in any number (zero or more) of PCR types.
4. A record type that does not participate as parent record type in any PCR type is called a **leaf** of the hierarchical schema.
5. If a record type participates as parent in more than one PCR type, then *its child record types are ordered*. The order is displayed, by convention, from left to right in a hierarchical diagram.

The definition of a hierarchical schema defines a **tree data structure**. In the terminology of tree data structures, a record type corresponds to a **node** of the tree, and a PCR type corresponds to an **edge** (or **arc**) of the tree. We use the terms *node* and *record type*, and *edge* and *PCR type*, interchangeably. The usual convention of displaying a tree is slightly different from that used in hierarchical diagrams, in that each tree edge is shown separately from other edges (Figure D.3). In hierarchical diagrams the convention is that all edges emanating from the same parent node are joined together (as in Figure D.1). We use this latter hierarchical diagram convention.

The preceding properties of a hierarchical schema mean that every node except the root has exactly one parent node. However, a node can have several child nodes, and in

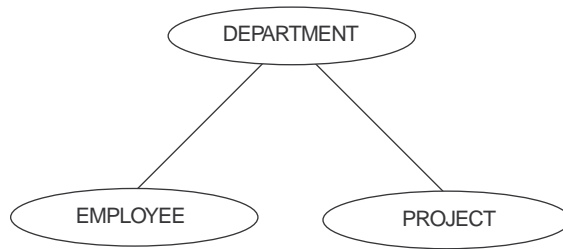


Figure D.3 A tree representation of the hierarchical schema in Figure D.1.

this case they are ordered from left to right. In Figure D.1 *EMPLOYEE* is the first child of *DEPARTMENT*, and *PROJECT* is the second child. The previously identified properties also limit the types of relationships that can be represented in a hierarchical schema. In particular, M:N relationships between record types *cannot* be directly represented, because parent-child relationships are 1:N relationships, and a record type *cannot participate as child* in two or more distinct parent-child relationships.

An M:N relationship may be handled in the hierarchical model by allowing duplication of child record instances. For example, consider an M:N relationship between *EMPLOYEE* and *PROJECT*, where a project can have several employees working on it, and an employee can work on several projects. We can represent the relationship as a (*PROJECT*, *EMPLOYEE*) PCR type. In this case a record describing the same employee can be duplicated by appearing once under *each* project that the employee works for. Alternatively, we can represent the relationship as an (*EMPLOYEE*, *PROJECT*) PCR type, in which case project records may be duplicated.

EXAMPLE 1: Consider the following instances of the *EMPLOYEE:PROJECT* relationship:

Project	Employees Working on the Project
A	E1, E3, E5
B	E2, E4, E6
C	E1, E4
D	E2, E3, E4, E5

If these instances are stored using the hierarchical schema (*PROJECT*, *EMPLOYEE*) (with *PROJECT* as the parent), there will be four occurrences of the (*PROJECT*, *EMPLOYEE*) PCR type—one for each project. The employee records for E1, E2, E3, and E5 will appear *twice each* as child records, however, because each of these employees works on two projects. The employee record for E4 will appear three times—once under each of projects B, C,

and D and may have number of hours that E4 works on each project in the corresponding instance.

To avoid such duplication, a technique is used whereby several hierarchical schemas can be specified in the same hierarchical database schema. Relationships like the preceding PCR type can now be defined across different hierarchical schemas. This technique, called **virtual relationships**, causes a departure from the “strict” hierarchical model. We discuss this technique in Section D.2.

D.1.3 Hierarchical Occurrence Trees

Corresponding to a hierarchical schema, many **hierarchical occurrences**, also called **occurrence trees**, exist in the database. Each one is a **tree structure** whose root is a single record from the root record type. The occurrence tree also contains all the children record occurrences of the root record and continues all the way to records of the leaf record types.

For example, consider the hierarchical diagram shown in Figure D.4, which represents part of the COMPANY database introduced in Chapter 3 and also used in Chapters 7, 8, and 9. Figure D.5 shows one hierarchical occurrence tree of this hierarchical schema. In the occurrence tree, each **node** is a **record occurrence**, and each arc represents a parent-child relationship between two records. In both Figures D.4 and D.5, we use the characters **D**, **E**, **P**, **T**, **S**, and **W** to represent **type indicators** for the record types DEPARTMENT, EMPLOYEE, PROJECT, DEPENDENT, SUPERVISEE, and WORKER, respectively. A node N and all its descendent nodes form a **subtree** of node N. An **occurrence tree** can be defined as the subtree of a record whose type is of the root record type.

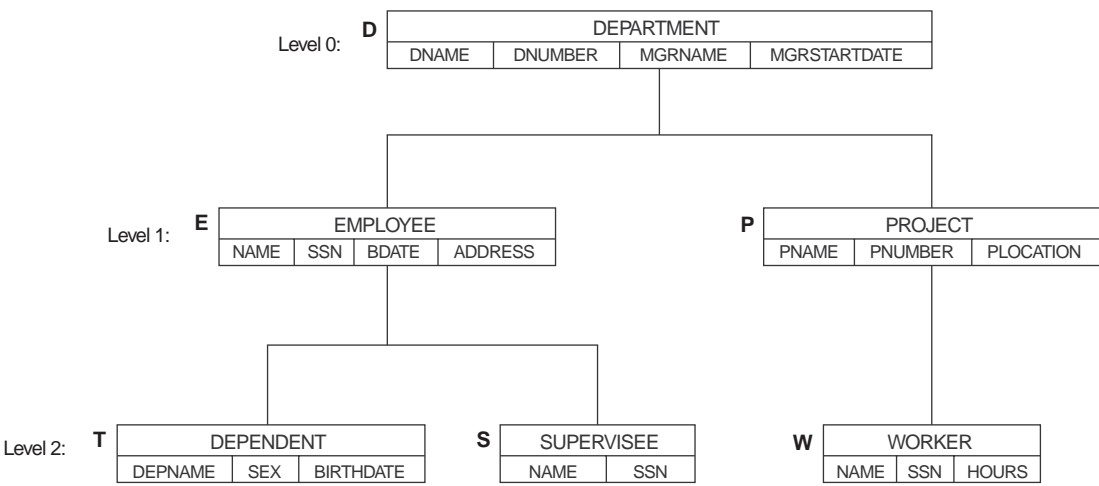


Figure D.4 A hierarchical schema for part of the COMPANY database.

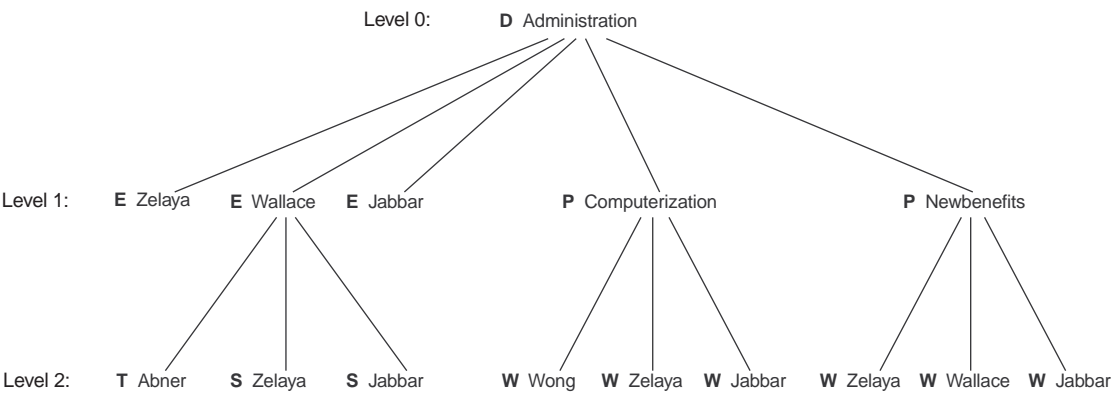


Figure D.5 An occurrence tree of the schema in Figure D.4.

D.1.4 Linearized Form of a Hierarchical Occurrence Tree

A hierarchical occurrence tree can be represented in storage by using any of a variety of data structures. However, a particularly simple storage structure that can be used is the **hierarchical record**, which is a linear ordering of the records in an occurrence tree in the **preorder traversal** of the tree. This order produces a sequence of record occurrences known as the **hierarchical sequence** (or **hierarchical record sequence**) of the occurrence tree; it can be obtained by applying a recursive procedure called the **pre-order traversal**, which visits nodes depth first and in a left-to-right fashion.

The occurrence tree in Figure D.5 gives the hierarchical sequence shown in Figure D.6. The system stores the type indicator with each record so that the record can be distinguished within the hierarchical sequence. The hierarchical sequence is also important because hierarchical data-manipulation languages, such as that used in IMS, use it as a basis for defining hierarchical database operations. The HDML language we discuss in

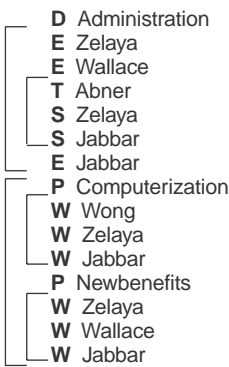


Figure D.6 Hierarchical sequence for the occurrence tree in Figure D.5.

Section D.3 (which is a simplified version of DL/1 of IMS) is based on the hierarchical sequence. A **hierarchical path** is a sequence of nodes N_1, N_2, \dots, N_i , where N_1 is the root of a tree and N_j is a child of N_{j-1} for $j = 2, 3, \dots, i$. A hierarchical path can be defined either on a hierarchical schema or on an occurrence tree. We can now define a **hierarchical database occurrence** as a sequence of all the occurrence trees that are occurrences of a hierarchical schema. For example, a hierarchical database occurrence of the hierarchical schema shown in Figure D.4 would consist of a number of occurrence trees similar to the one shown in Figure D.5, one for each distinct department.

D.1.5 Virtual Parent-Child Relationships

The hierarchical model has problems when modeling certain types of relationships. These include the following relationships and situations:

1. M:N relationships.
2. The case where a record type participates as child in more than one PCR type.
3. N-ary relationships with more than two participating record types.

Notice that the relationship between EMPLOYEE and EPOINTER in Figure D.7(a) is a 1:N relationship and hence qualifies as a PCR type. Such a relationship is called a **virtual parent-child relationship (VPCR) type**.² EMPLOYEE is called the **virtual parent** of EPOINTER; and conversely, EPOINTER is called a **virtual child** of EMPLOYEE. Conceptually, PCR types and VPCR types are similar. The main difference between the two lies in the way they are implemented. A PCR type is usually implemented by using the hierarchical

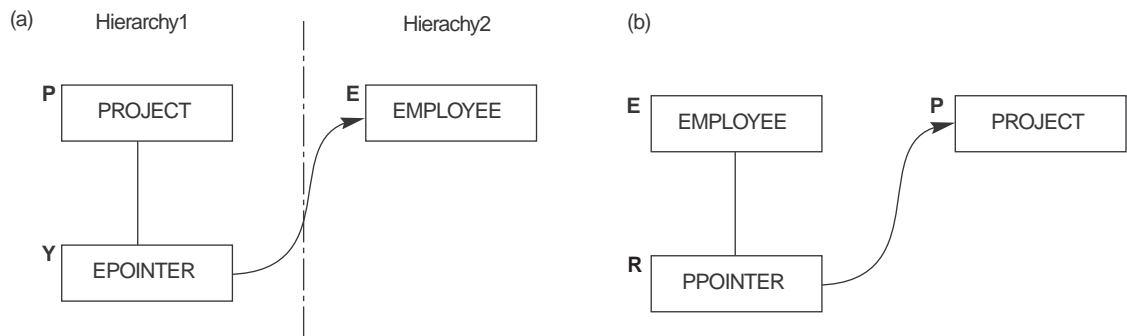


Figure D.7 Representing M:N relationships using Virtual Parent Child Relationships. (a) EMPLOYEE as virtual parent. (b) PROJECT as virtual parent.

2. The term “virtual” is not used in the IMS system, but it is used here to simplify the distinction between hierarchical relationships within one hierarchy (called Physical in IMS) and across hierarchies (called Logical in IMS).

sequence, whereas a VPCR type is usually implemented by establishing a pointer (a physical one containing an address, or a logical one containing a key) from a virtual child record to its virtual parent record. This mainly affects the efficiency of certain queries.

Figure D.8 shows a hierarchical database schema of the COMPANY database that uses some VPCRs and has no redundancy in its record occurrences. The hierarchical database schema is made up of two hierarchical schemas—one with root DEPARTMENT, and the other with root EMPLOYEE. Four VPCRs, all with virtual parent EMPLOYEE, are included to represent the relationships without redundancy. Notice that IMS *may not allow this* because an implementation constraint in IMS limits a record to being virtual parent of at most one VPCR; to get around this constraint, one can create dummy children record types of EMPLOYEE in Hierarchy 2 so that each VPCR points to a distinct virtual parent record type.

In general, there are many feasible methods of designing a database using the hierarchical model. In many cases, performance considerations are the most important factor in choosing one hierarchical database schema over another. Performance depends

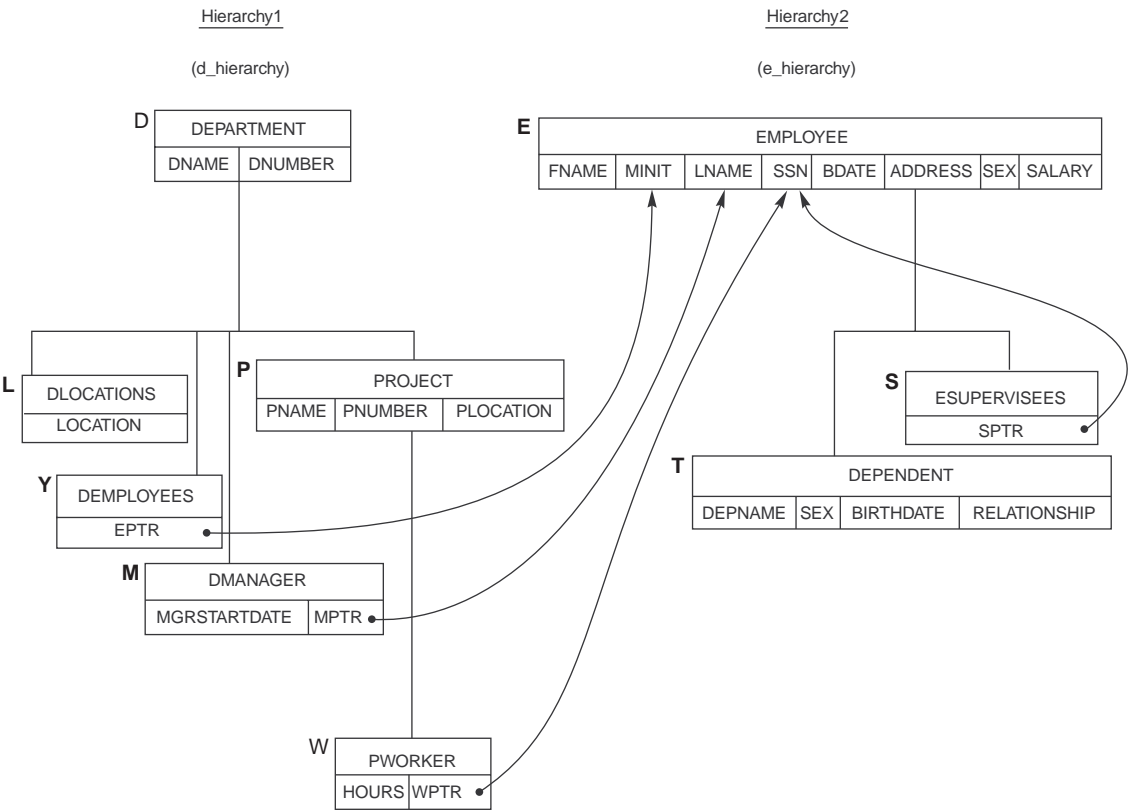


Figure D.8 Using VPCR types to eliminate redundancy in the COMPANY database.

on the implementation options available—for example, whether certain types of pointers are provided by the system and whether certain limits on number of levels are imposed by the DBA.

D.2 Integrity Constraints and Data Definition in the Hierarchical Model

D.2.1 Integrity Constraints in the Hierarchical Model

A number of built-in **inherent constraints** exist in the hierarchical model whenever we specify a hierarchical schema. These include the following constraints:

1. No record occurrences except root records can exist without being related to a parent record occurrence. This has the following implications:
 - a. A child record cannot be inserted unless it is linked to a parent record.
 - b. A child record may be deleted independently of its parent; however, deletion of a parent record automatically results in deletion of all its child and descendant records.
 - c. The above rules do not apply to virtual child records and virtual parent records.
2. If a child record has two or more parent records from the *same* record type, the child record must be duplicated once under each parent record.
3. A child record having two or more parent records of *different* record types can do so only by having at most one real parent, with all the others represented as virtual parents. IMS limits the number of virtual parents to one.
4. In IMS, a record type can be the virtual parent in *only one* VPCR type. That is, the number of virtual children can be only one per record type in IMS.

D.2.2 Data Definition in the Hierarchical Model

In this section we give an example of a hierarchical data definition language (HDDL), which is *not* the language of any specific hierarchical DBMS but is used to illustrate the language concepts for a hierarchical database. The HDDL demonstrates how a hierarchical database schema can be defined. To define a hierarchical database schema, we must define the fields of each record type, the data type of each field, and any key constraints on fields. In addition, we must specify a root record type as such; and for every nonroot record type, we must specify its (real) parent in a PCR type. Any VPCR types must also be specified.

In Figure D.9, either each record type is declared to be of type root or a single (real) parent record type is declared for the record type. The data items of the record are then listed along with their data types. We must specify a virtual parent for data items that are of type *pointer*. Data items declared under the KEY clause are constrained to have unique values for each record. Each KEY clause specifies a separate key; in addition, if a single KEY clause lists more than one field, the combination of these field values must be unique in

SCHEMA NAME = COMPANY

HIERARCHIES = HIERARCHY1, HIERARCHY2

RECORD

NAME = EMPLOYEE
 TYPE = ROOT OF HIERARCHY2
 DATA ITEMS =
 FNAME CHARACTER 15
 MINIT CHARACTER 1
 LNAME CHARACTER 15
 SSN CHARACTER 9
 BDATE CHARACTER 9
 ADDRESS CHARACTER 30
 SEX CHARACTER 1
 SALARY CHARACTER 10
 KEY = SSN
 ORDER BY LNAME, FNAME

RECORD

NAME = DEPARTMENT
 TYPE = ROOT OF HIERARCHY1
 DATA ITEMS =
 DNAME CHARACTER 15
 DNUMBER INTEGER
 KEY = DNAME
 KEY = DNUMBER
 ORDER BY DNAME

RECORD

NAME = DLOCATIONS
 PARENT = DEPARTMENT
 CHILD NUMBER = 1
 DATA ITEMS =
 LOCATION CHARACTER 15

RECORD

NAME = DMANAGER
 PARENT = DEPARTMENT
 CHILD NUMBER = 3
 DATA ITEMS =
 MGRSTARTDATE CHARACTER 9
 MPTR POINTER WITH VIRTUAL PARENT = EMPLOYEE

RECORD

NAME = PROJECT
 PARENT = DEPARTMENT
 CHILD NUMBER = 4
 DATA ITEMS =
 PNAME CHARACTER 15
 PNUMBER INTEGER
 PLOCATION CHARACTER 15
 KEY = PNAME
 KEY = PNUMBER
 ORDER BY PNAME

RECORD

NAME = PWORKER
 PARENT = PROJECT
 CHILD NUMBER = 1
 DATA ITEMS =
 HOURS CHARACTER 4
 WPTR POINTER WITH VIRTUAL PARENT = EMPLOYEE

Figure D.9 HDDL declarations for the hierarchical schema in Figure D.8.

```

RECORD
  NAME = DEMPLOYEES
  PARENT = DEPARTMENT
  CHILD NUMBER = 2
  DATA ITEMS =
    EPTR      POINTER WITH VIRTUAL PARENT = EMPLOYEE

RECORD
  NAME IS ESUPERVISEES
  PARENT = EMPLOYEE
  CHILD NUMBER = 2
  DATA ITEMS =
    SPTR      POINTER WITH VIRTUAL PARENT = EMPLOYEE

RECORD
  NAME = DEPENDENT
  PARENT = EMPLOYEE
  CHILD NUMBER = 1
  DATA ITEMS =
    DEPNAME    CHARACTER 15
    SEX        CHARACTER 1
    BIRTHDATE  CHARACTER 9
    RELATIONSHIP CHARACTER 10
  ORDER BY DESC BIRTHDATE

```

Figure D.9 (Continued)

each record. The CHILD NUMBER clause specifies the left-to-right order of a child record type under its (real) parent record type. The ORDER BY clause specifies the order of individual records of the same record type in the hierarchical sequence. For nonroot record types, the ORDER BY clause specifies how the records should be ordered *within each parent record*, by specifying a field called a **sequence key**. For example, PROJECT records controlled by a particular DEPARTMENT have their subtrees ordered alphabetically within the same parent DEPARTMENT record by PNAME, according to Figure D.9.

D.3 Data Manipulation Language for the Hierarchical Model

We now discuss **Hierarchical Data Manipulation Language (HDML)**, which is a record-at-a-time language for manipulating hierarchical databases. We have based its structure on IMS's DL/1 language. It is introduced to illustrate the concepts of a hierarchical database manipulation language. The commands of the language must be embedded in a general-purpose programming language called the **host language**.

The HDML is based on the concept of **hierarchical sequence** defined in Section D.1. Following each database command, the last record accessed by the command is called the **current database record**. The DBMS maintains a pointer to the current record. Subsequent database commands *proceed from the current record* and may define a new current record, depending on the type of command.

D.3.1 The GET Command

The HDML command for retrieving a record is the **GET command**. There are many variations of GET; the structure of two of these variations is as follows, with optional parts enclosed in brackets [...]:

- **GET FIRST**³ <record type name> [WHERE <condition>]
- **GET NEXT** <record type name> [WHERE <condition>]

The simplest variation is the **GET FIRST command**, which always starts searching the database from the *beginning of the hierarchical sequence* until it finds the first record occurrence of <record type name> that satisfies <condition>. This record also becomes the current of database, current of hierarchy, and current of record type and is retrieved into the corresponding program variable. For example, to retrieve the “first” EMPLOYEE record in the hierarchical sequence whose name is John Smith, we write EX1:

```
EX1: $GET FIRST EMPLOYEE WHERE FNAME = 'John' AND LNAME = 'Smith';
```

The DBMS uses the condition following WHERE to search for the first record in order of the hierarchical sequence that satisfies the condition and is of the specified record type. If more than one record in the database satisfies the WHERE condition and we want to retrieve all of them, we must write a looping construct in the host program and use the GET NEXT command. We assume that the GET NEXT starts its search from the *current record of the record type specified in GET NEXT*,⁴ and it searches forward in the hierarchical sequence to find another record of the specified type satisfying the WHERE condition. For example, to retrieve records of all EMPLOYEES whose salary is less than \$20,000 and obtain a printout of their names, we can write the program segment shown in EX2:

```
EX2: $GET FIRST EMPLOYEE WHERE SALARY < '20000.00';
      while DB_STATUS = 0 do
        begin
          writeln (P_EMPLOYEE.FNAME, P_EMPLOYEE.LNAME);
          $GET NEXT EMPLOYEE WHERE SALARY < '20000.00'
        end;
```

In EX2, the while loop continues until no more EMPLOYEE records in the database satisfy the WHERE condition; hence, the search goes through to the last record in the database (hierarchical sequence). When no more records are found, DB_STATUS becomes nonzero, with a code indicating “end of database reached,” and the while loop terminates.

D.3.2 The GET PATH and GET NEXT WITHIN PARENT Retrieval Commands

So far we have considered retrieving single records by using the GET command. But when we have to locate a record deep in the hierarchy, the retrieval may be based on a series of

3. This is similar to the GET UNIQUE (GU) command of IMS.

4. IMS commands generally proceed forward from the *current of database*, rather than from the current of specified record type as HDML commands do.

conditions on records along the entire hierarchical path. To accommodate this, we introduce the GET PATH command:

GET (FIRST | NEXT) PATH <hierarchical path> [**WHERE** <condition>]

Here, <hierarchical path> is a list of record types that starts from the root along a path in the hierarchical schema, and <condition> is a Boolean expression specifying conditions on the individual record types along the path. Because several record types may be specified, the field names are prefixed by the record type names in <condition>. For example, consider the following query: “List the lastname and birthdates of all employee-dependent pairs, where both have the first name John.” This is shown in EX3:

```
EX3: $GET FIRST PATH EMPLOYEE, DEPENDENT
WHERE EMPLOYEE.FNAME='John' AND DEPENDENT.DEPNAME='John';
while DB_STATUS = 0 do
  begin
    writeln (P_EMPLOYEE.FNAME, P_EMPLOYEE.BDATE,
            P_DEPENDENT.BIRTHDATE);
    $GET NEXT PATH EMPLOYEE, DEPENDENT
      WHERE EMPLOYEE.FNAME='John' AND
        DEPENDENT.DEPNAME='John'
  end;
```

We assume that a GET PATH command retrieves *all records along the specified path* into the user work area variables,⁵ and the last record along the path becomes the current database record. In addition, all records along the path become the current records of their respective record types.

Another common type of query is to find all records of a given type that have *the same parent record*. In this case we need the GET NEXT WITHIN PARENT command, which can be used to loop through the child records of a parent record and has the following format:

GET NEXT <child record type name>
WITHIN [VIRTUAL] PARENT [<parent record type name>]⁶
 [**WHERE** <condition>]

This command retrieves the next record of the child record type by searching forward from the current of the child record type for the next child record owned by the current parent record. If no more child records are found, DB_STATUS is set to a nonzero value to indicate that “there are no more records of the specified child record type that have the same parent as the current parent record.” The <parent record type name> is *optional*, and the default is the immediate (real) parent record type of <child record type name>. For example, to retrieve the names of all projects controlled by the ‘Research’ department, we can write the program segment shown in EX4:

```
EX4: $GET FIRST PATH DEPARTMENT, PROJECT
WHERE DNAME ='Research';
```

5. IMS provides the capability of specifying that only *some* of the records along the path are to be retrieved.

6. There is no provision for retrieving all children of a virtual parent in IMS in this way without defining a view of the database.

```
(* the above establishes the 'Research' DEPARTMENT record as
   current parent of type DEPARTMENT, and retrieves the first
   child PROJECT record under that DEPARTMENT record *)
while DB_STATUS = 0 do
  begin
    writeln (P_PROJECT.PNAME);
    $GET NEXT PROJECT WITHIN PARENT
  end;
```

D.3.3 HDML Commands for Update

The HDML commands for updating a hierarchical database are shown in Table D.1, along with the retrieval command. The **INSERT** command is used to insert a new record. Before inserting a record of a particular record type, we must first place the field values of the new record in the appropriate user work area program variable.

The **INSERT** command inserts a record into the database. The newly inserted record also becomes the current record for the database, its hierarchical schema, and its record type. If it is a root record, as in EX8, it creates a new hierarchical occurrence tree with the new record as root. The record is inserted in the hierarchical sequence in the order specified by any **ORDER BY** fields in the schema definition.

To insert a child record, we should make its parent, or one of its sibling records, the *current record* of the hierarchical schema before issuing the **INSERT** command. We should also set any virtual parent pointers before inserting the record.

To delete a record from the database, we first make it the current record and then issue the **DELETE** command. The **GET HOLD** is used to make the record the current record, where the **HOLD** key word indicates to the DBMS that the program will delete or

Table D.1 Summary of HDML Commands

RETRIEVAL	
GET	Retrieve a record into the corresponding program variable and make it the current record. Variations include GET FIRST, GET NEXT, GET NEXT WITHIN PARENT, and GET PATH.
RECORD UPDATE	
INSERT	Store a new record in the database and make it the current record.
DELETE	Delete the current record (and its subtree) from the database.
REPLACE	Modify some fields of the current record.
CURRENCY RETENTION	
GET HOLD	Retrieve a record and hold it as the current record so it can subsequently be deleted or replaced.

update the record just retrieved. For example, to delete all male EMPLOYEES, we can use EX5, which also lists the deleted employee names <before> deleting their records:

```
EX5:  SGET HOLD FIRST EMPLOYEE WHERE SEX='M';
      while DB_STATUS=0 then
        begin
          writeln (P_EMPLOYEE.LNAME, P_EMPLOYEE.FNAME);
          SDELETE EMPLOYEE;
          SGET HOLD NEXT EMPLOYEE WHERE SEX='M';
        end;
```

D.3.4 IMS—A Hierarchical DBMS

IMS is one of the earliest DBMSs, and it ranks as the dominant system in the commercial market for support of large-scale accounting and inventory systems. IBM manuals refer to the full product as IMS/VS (Virtual Storage), and typically the full product is installed under the MVS operating system. IMS DB/DC is the term used for installations that utilize the product's own subsystems to support the physical database (DB) and to provide data communications (C).

However, other important versions exist that support only the IMS data language—Data Language One (DL/1). Such DL/1-only configurations can be implemented under MVS, but they may also use the DOS/VSE operating system. These systems issue their calls to VSAM files and use IBM's Customer Information Control System (CICS) for data communications. The trade-off is a sacrifice of support features for the sake of simplicity and improved throughput.

A number of versions of IMS have been marketed to work with various IBM operating systems, including (among the recent systems) OS/VS1, OS/VS2, MVS, MVS/XA, and ESA. The system comes with various options. IMS runs under different versions on the IBM 370 and 30XX family of computers. The data definition and manipulation language of IMS is DL/1. Application programs written in COBOL, PL/1, FORTRAN, and BAL (Basic Assembly Language) interface with DL/1.

Selected Bibliography

The first hierarchical DBMS—IMS and its DL/1 language—was developed by IBM and North American Aviation (Rockwell International) in the late 1960s. Few early documents exist that describe IMS. McGee (1977) gives an overview of IMS in an issue of IBM Systems Journal devoted to ims. Bjoerner and Lovengren (1982) formalize some aspects of the IMS data model. Kapp and Leben (1986) is a popular book on IMS programming. IMS is described in a very large collection of IBM manuals.

Recent work has attempted to incorporate hierarchical structures in the relational model (Gyssens et al., 1989; Jagadish, 1989). This includes nested relational models (see Section 13.6).